Service-Oriented Architecture Modeling: Bridging the Gap between Structure and Behavior

Mickaël Clavreul, Sébastien Mosser, Mireille Blay-Fornarino, Robert B. France

Service–Oriented Architecture Modeling: Bridging the Gap between Structure and Behavior

Mickael Clavreul[1], Sébastien Mosser[2],
Mireille Blay–Fornarino[3], and Robert B. France[4]

[1] INRIA, Campus Universitaire de Beaulieu, 35042 Rennes, France
mickael.clavreul@inria.fr
[2] INRIA Lille–Nord Europe, LIFL (UMR CNRS 8070), Univ. Lille 1, France
sebastien.mosser@inria.fr
[3] I3S (UMR CNRS 6070), Université Nice–Sophia Antipolis, France
blay@polytech.unice.fr
[4] Colorado State University, Fort Collins, CO, USA
france@cs.colostate.edu

Abstract. Model–driven development of large-scale software systems is highly likely to produce models that describe the systems from many diverse perspectives using a variety of modeling languages. Checking and maintaining consistency of information captured in such multi-modeling environments is known to be challenging. In this paper we describe an approach to systematically synchronize multi–models. The approach specifically addresses the problem of synchronizing business processes and domain models in a Service-oriented Architecture development environment. In the approach, the human effort required to synchronize independently developed models is supplemented with significant automated support. This process is used to identify concept divergences, that is, a concept in one model which cannot be matched with concepts in the other model. We automate the propagation of divergence resolution decisions across the conflicting models. We illustrate the approach using models developed for a Car Crash Crisis Management System (CCCMS), a case study problem used to assess Aspect–oriented Modeling approaches.

1 Introduction

Developing a large–scale software system as a Service–oriented Architecture (SOA) involves the creation and integration of a variety of services. Services must be coordinated to adequately participate in the required behavior of the system. Model–driven development of such systems is highly likely to produce a variety of models capturing the many diverse design concerns that arise during development. The management of models in such multi–modeling environments is known to be challenging. In particular, activities related to checking and maintaining consistency among the multiple views of a system can be complex. There is a need for techniques that developers can use to detect conflicts and divergences across multi-models of systems developed using SOA. Two models diverge when one model consists of elements that do not correspond to elements in the other model.

J. Whittle, T. Clark, and T. Kühne (Eds.): MODELS 2011, LNCS 6981, pp. 289–303, 2011.

Our work specifically addresses the problem of synchronizing SOA business process models with domain models. The approach described in this paper provides SOA designers with integrated generative and model composition techniques that can be used to automatically propagate divergence resolution strategies across these models. The core of the iterative synchronization approach consists of four major steps: *(i)* the generation of a structural model based on the data extracted from the business process model, *(ii)* the merge of the generated model with the initial domain model, *(iii)* the identification of formal divergences between these two models and finally *(iv)* the automated propagation of resolution strategies provided by experts.

The remainder of this paper is organized as follows. Section 2 introduces the CCCMS case study that motivates our approach. Section 3 outlines the challenges and the solution that we propose in this paper. Section 4 presents situations where divergences occur and proposes a formalization of the divergences. Section 5 illustrates how we capture experts knowledge about how to resolve divergences. Section 6 focuses on the fourth step of the process and describes how resolution strategies are automatically propagated across both the domain model and the business processes model. Section 7 discusses related work and Section 8 concludes this paper.

2 Car Crash Crisis Management System (CCCMS)

We illustrate the approach using a case study problem described in a Transactions on Aspect-Oriented Software Development (TAOSD) special issue on Aspect-Oriented Modeling (AOM) [15]. The purpose of the special issue was to compare the application of existing AOM approaches on a common system development problem, namely the development of a Crisis Management System (CMS). In the case study, a CMS is *"a system that facilitates coordination of activities and information flow between all stakeholders and parties that need to work together to handle a crisis"* [11]. Among the multitude of crises handled by CMS, including terrorist attacks, epidemics, or accidents, we focus on car accidents. Car accidents are handled by the Car Crash CMS (CCCMS) which *"includes all the functionalities of general crisis management systems, and some additional features specific to car crashes such as facilitating the rescuing of victims at the crisis scene and the use of tow trucks to remove damaged vehicles"*. The original system includes ten use cases described using textual scenarios.

For ease of understanding, we illustrate our approach on the *Capture Witness Report (CWR)* use case only. The CWR case study (use case #2 in the original document) captures the set of actions that a *Coordinator* takes to create a new *Crisis* based on the information reported by the *Witness* of a car accident. The main success scenario for this use case (extracted from the requirements document) is described in FIG. 1. The subject of the use case is the CCCMS system represented by *System*. Two actors are involved in the sequence of activities needed to report a car crash: *(i) PhoneCompany* is the role played by an external partner that provides phone–related information, and *(ii) Coordinator* is the role played by the person who interacts with the CCCMS system through a graphical user interface to enter information.

We focus on the contribution of two experts in the definition of a solution to this CWR use case: a domain model expert (e_d) designs the structural view of the system

Coordinator requests *Witness* to provide his identification.
1. *Coordinator* provides witness information to *System* as reported by the witness.
2. *Coordinator* informs *System* of location and type of crisis as reported by the witness.
   In parallel to steps 2 – 4:
   2a.1 *System* contacts *PhoneCompany* to verify witness information.
   2a.2 *PhoneCompany* sends address/phone information to *System*.
   2a.3 *System* validates information received from the *PhoneCompany*.
3. *System* provides *Coordinator* with a crisis-focused checklist.
4. *Coordinator* provides crisis information to *System* as reported by the witness.
5. *System* assigns an initial emergency level to the crisis and sets the crisis status to active.

Use case ends in success.

Fig. 1. Textual Scenario of Use Case #2: "Capture Witness Report"

and a business process expert (e_b) designs the behavioral view (*i.e.*, the set of activities and the flow of control between these activities) of the system.

Domain Model Design. FIG. 2(a) is a class diagram that captures problem concepts identified from the requirements and that are relevant to the CWR use case. This domain class diagram (CD_D) is designed by e_d who formalizes his deep understanding of the various concepts manipulated in the CCCMS system. The main concepts with respect to the CWR use case are the following:

`Crisis`: is the concept shared by any CMS system. A **`Crisis`** occurs at a given location and at a given time, it has an emergency level, a status and possibly some additional information. A **`Crisis`** may be reported by a **`Witness`** and may include **`Mission`**s.

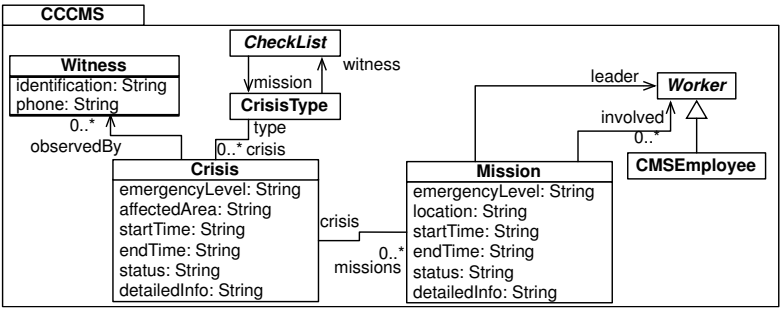
`Witness`: is a person who reports a **`Crisis`**.

`Mission`: is an action that should be taken when a **`Crisis`** is reported.

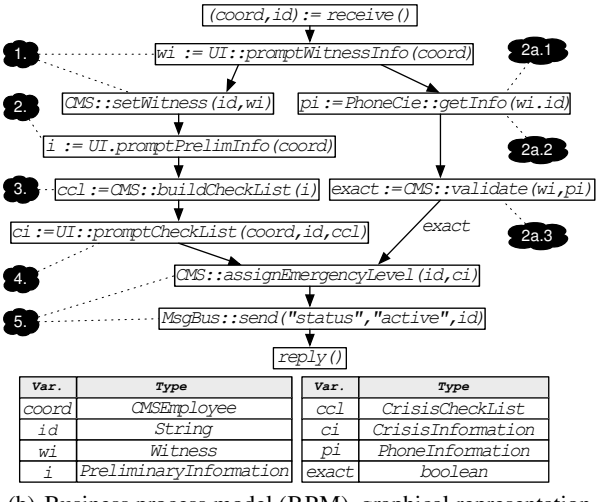
`CheckList`: is a list of things that should be checked with a **`Witness`**.

`CMSEmployee`: is a human resource who is qualified and capable of performing **`Mission`**s in the context of a **`Crisis`**.

Business Process Model. The business process model (BPM) associated with the CWR use case is represented in FIG. 2(b). According to SOA principles, e_b designs this business process model with regard to his/her own understanding of the system. For better undestanding, we provide correspondences (black clouds) between the BPM activities and the steps in the textual scenario (see Fig. 1). The business process starts by receiving a crisis coordinator (**`coord`**) and a crisis identifier (**`id`**). It contains two branches, executed in parallel. The left branch of the business process deals with the internal logic of the CWR scenario. The context of the current crisis is built by retrieving information from the witness of the crisis: the process requests preliminary information about the crisis and then refines the information it receives through subsequent exchanges between the system and the witness. In parallel (the right branch), the system calls an external partner (**`PhoneCompany`**) to check the information given by the witness of a crisis and prevent false or erroneous reports. When the two branches join, that is, when the system considers the crisis report to be genuine, the system assigns an emergency level to the crisis and updates the crisis status to **`active`**.



(a) Structural model (CD_D), extract.



(b) Business process model (BPM), graphical representation

We use here the graphical representation defined by ADORE *[15] to represent business processes. Boxes represent activities (*e.g.*, message reception, service invocation), and arrows represent causality relations (*i.e.*, the associated partial order). A wait relation (*$\texttt{a} \rightarrow \texttt{b}$*) means that* b *will wait for the end of* a *to start its own execution. A guard relation (*$\texttt{a} \xrightarrow{\texttt{v}} \texttt{b}$*) strengthens the wait semantics, and conditions the start of* b *to the value of* v*. Relations are combined using a conjunctive semantics* ($\wedge$).

Fig. 2. Initial model artifacts, proposed by experts

3 Challenges and Synchronization Process

The complete CCCMS implementation contains thirteen business processes, describing hundreds of activities and thousands of relations between activities. Manual synchronization of the various views of such a large system can be challenging, time–consuming and error–prone. This section highlights situations in which checking and maintaining consistency across models can benefit from the use of automatic synchronization mechanisms. Since CD_D and BPM are defined by independent experts ($e_d \neq e_b$), one can encounter situations where types from the behavioral model (BPM) and types from the structural model (CD_D) diverge. We illustrate these divergences with examples from Section 2 below:

S_1**–Name Mismatch:** The business expert misspells a concept that already exists in CD_D. In FIG. 2(b), e_d uses a **`CheckList`** type whereas e_b uses a **`CrisisCheckList`** type. This situation illustrates naming conflicts that often occur across different views of the same system. For instance, the PROMPT [17] approach for aligning ontologies addresses this kind of conflicts among others.

S_2**–Concept Enforcing:** The business expert uses data collected from an external partner, which are unknown from the domain point of view. In FIG. 2(b), e_b uses information collected from the external agency *PhoneCompany* that is unkown to e_d and thus not modeled in the CD_D. This situation identifies the need to introduce externally defined artefacts (*i.e.*, provided by partner services) to the CD_D.

S_3**–Concept Usages:** The business expert uses his/her own data structure, *i.e.*, uses concepts defined in CD_D in an unforeseen way. In FIG. 2(b), e_b uses a **`PreliminaryInformation`** concept in Activity 2. Since the original scenario indicates that the **`Coordinator`** should manipulate the location and type of the **`Crisis`**, we consider that e_b aggregated several artifacts already defined in CD_D (namely the location of the crisis and its type) in a single object for practical reasons. This situation illustrates how specific usage of data in a BPM can improve the CD_D.

Clearly, the synchronization of both CD_D and BPM is not a trivial problem. We identify two challenges related to these situations: *(i)* the automatic identification of such divergences (C_1) and *(ii)* the capture of resolution strategies and their automated propagation across models in the synchronization process (C_2). FIG. 3 illustrates our approach that tackles these two challenges. The first step of the process extracts data from the set of available BPM to derive a class diagram (CD_I) which contains all the concepts manipulated by this set of processes (1). Then, we use a *divergence detection* algorithm to identify occurrences of the situations (S_i) that we discussed previously (2). The detection of divergences leads to a phase of negotiation between experts from the domain and experts from the business process. Experts should consent on identifying *strategies* to resolve divergences (3) and to ultimately perform an accurate synchronization of CD_D and BPM. The last step of the process (4) propagates the resolution strategies using a dedicated algorithm (*strategies propagation*), which automatically applies changes in both CD_D and BPM.

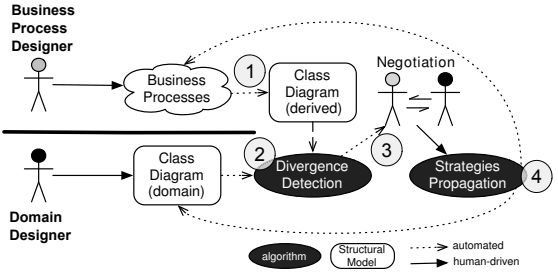


Fig. 3. SOA Models Synchronization: Process Overview

4 Identifying Model Divergences

This section presents the first two steps of the model synchronization process and the formalization of the divergence detection mechanism.

4.1 Naive Synchronization with Merge

The first step of the process extracts data from the BPM to derive a class–diagram (CD_I). The generation procedure visits all available business processes and extracts the types of all the declared variables.

Merging CD_I with CD_D using model composition techniques such as Kompose [8], produces a naive alignment of both models (FIG. 4). Naive alignment relies on an element matching process based on names. Elements with equivalent names are unified into a single element. For instance, the **`CMSEmployee`** element has been found in both CD_D and CD_I and therefore the merged model contains a single unified **`CMSEmployee`** element. Though simple, the naive alignment cannot align concepts that have different names. The default behavior of Kompose when such name–mismatches occur is to include the elements that do not match in the merged model. For instance, **`Preliminary-Information`** is a concept from CD_I with no candidate match in CD_D.

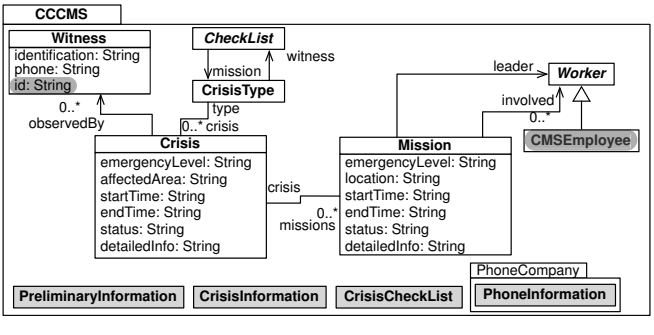


Fig. 4. Merged model: CD_D (white) $\oplus$ CD_I (gray)

We modified the default behavior of Kompose to record every operation used to produce the merged model. This record is analyzed to (1) validate every element that is automatically merged (*e.g.*, **`CMSEmployee`**) and to (2) detect divergences between CD_D and CD_I.

4.2 Intuitive Definition of Divergences

The analysis of the recorded operations leads to the detection of two kinds of divergences:

Point-of-view divergences occur when a model element from CD_I has no equivalent counterpart in CD_D (*e.g.*,**`PhoneInformation`**).

Structural divergences occur when a model element from CD_I has an equivalent counterpart in CD_D but the properties of the model element do not match with the properties of the corresponding model element in CD_D (*e.g.*, a "public" model element in CD_I is "private" in CD_D).

4.3 Divergence Detection Formalization

The divergence detection mechanism uses a matching operator and a set of signatures to compare a model element with another one. Let *match* be the predicate that checks if a model element of CD_I is equivalent to a model element of CD_D. With this match predicate, we formalize the kind of divergences as follows:

- *Point-of-view Divergence* refers to a model element in CD_I that has no equivalent model element in CD_D: $b \in CD_I s.t.\ \nexists d_i \in CD_D,\ match(b,d_i)$.
- *Structural Divergence* refers to a model element in CD_I that has equivalent model element in CD_D but whose properties do not match.

We formalize structural divergences according to the definitions provided by Barais *et al.* [3]. We defined two rules, used to reify the Class signature and the Property signature.

Class Signature. The signature of a Class encompasses its $identifier$, its $modifier$, possible $superclass$es and its $usage$. In the Object–Oriented (OO) paradigm, the $category$ and the $visibility$ of classes provide additional information on how we may use these classes in a given OO program. A class is *internal* when it participates in calling internal services either as a value or as the type of a parameter of a service. For all other usages, we consider the class as *mixed*.

$$Class^{sig} = (Identifier, Modifier, Superclass, Usage)$$
$$Modifiers \in \{Category, Visibility\},\ Category \in \{abstract, concrete, final\}$$
$$Visibility \in \{private, protected, public\},\ Usage \in \{internal, mixed\}$$

CD_I reflects the usage of the class definitions at runtime and thus, classes are necessarily *concrete*, *public* with no *Superclass*es. In other words, we detect a divergence (c1) when a class in CD_I has an equivalent class in CD_D that is not *public*:

$$(c1)\ match(C_B, C_D) \wedge Visibility_{C_D} \neq public \tag{1}$$

Usage refers to the class usage in the business processes. This definition has an impact on the process of deriving CD_I: (1) classes that do not participate in calling an internal service are not captured by the data structure extraction process since we cannot modify the definition of a class provided by an external partner for compatibility reasons; (2) classes that are used both within internal and external services are *mixed*. They can only be enriched with additional information that cope with the initial definition of the class. Regarding *Usage*, we detect a divergence (c2) when the usage of a class in CD_D is internal whereas an equivalent class is *mixed* in CD_I:

$$(c2)\ match(C_B, C_D) \wedge Usage_{C_D} = internal \wedge Usage_{C_B} = mixed \tag{2}$$

Property Signature. The signature of a property encompasses its *Identifier*, its scope of use (*Static*), its *Type* that is either a *Class* or a *Datatype* and its *Access*.

$$\begin{aligned} Property^{sig} &= (Identifier, Static, Type, Access) \\ Static &\in \{static, nonstatic\},\ Type \in Class \cup Datatype \\ Access &\in \{read, write, rw, no\} \end{aligned}$$

The first divergence (p1) that we may detect is if the two properties that we matched in CD_I and in CD_D have different types:

$$match(P_B, P_D) \wedge (p1)\ Type_{P_D} \neq Type_{P_B} \tag{3}$$

A property is *static* if it is common to all instances of this property and it is *nonstatic* otherwise. Properties that are used in BPM are necessarily *nonstatic* and thus we may detect the following divergence (p2):

$$match(P_B, P_D) \wedge (p2)\ Static_{P_D} = static \tag{4}$$

Among these usual OO characteristics, we propose an additional *access* characteristic which determines how a property is accessed in BPM: *read* means that the property is only read by a service; *write* means that the property is only written by a service; *rw* means that the property is read and written by one or more services; *no* is used in other cases. For instance, the property *id* of a **`Witness`** in FIG. 2(b) is a *read* property since the property is read in activity `2a.1` and never written in any other activity. From this definition, we may detect two divergences: (p3) a property in CD_D is never accessed (*no*) or ($p3'$) a property in CD_D is not *rw* and an equivalent property in CD_I is accessed differently:

$$(p3)\ Access_{P_D} = no\ \vee (p3')\ (Access_{P_D} \neq rw \wedge Access_{P_D} \neq Access_{P_B}) \tag{5}$$

The formalization of the various kind of divergences allows the definition of generic resolution strategies that we discuss in the next section.

5 Resolution Strategies

This section proposes a formal representation of the resolution strategies (a graphical representation is presented in Fig. 5) to automate their propagation.

In the context of this paper, we focus on Point–of–View divergences, since their resolution requires action from humans and impacts both CD_D and BPM. Resolution of Point-of-View divergences involves a negotiation phase between the experts of the domain and the experts of the business process. Negotiation leads to a consensus on proposing a set of resolution strategies to properly synchronize CD_D with the data structure used in BPM.

To support the negotiation phase and to automate the propagation of resolution strategies, we propose a high-level specification of these resolution strategies, using a mapping language and the graphical tool that supports it. The mapping language and the tool are based on previous work [5]. In this specific case study, we map models of different views of the same system instead of expressing mapping on heterogeneous metamodels. The original definition of a mapping relationship remains: a mapping relationship is a white diamond which has links (dotted lines) to model elements from CD_D and CD_I.

The definition of a mapping strategy is slightly different from [5] since it depends on the types of elements involved in the mapping and the arity of the relationship (*i.e.*, the number of model elements involved in the mapping relationship). The meaning of mapping strategies is to ultimately align CD_I and CD_D data structures and we propose two unidirectional alignment strategies for synchronizing CD_D and BPM:

- *Similarity strategy* addresses the problem of name mismatch (S_1). This strategy allows renaming some classes or properties to allow matching. Experts choose the name of an element that they consider as correct and they expect that each occurrence of the inadequate name is replaced by the chosen name. In FIG. 5, experts chose to keep **`CrisisCheckList`** from CD_I instead of **`CheckList`** from CD_D. A similarity strategy must be bound to a mapping between exactly two (arity = 1) model elements of the same type.
- *Replacement strategy* is chosen by experts when they select which model element from CD_I or from CD_D to keep when addressing the two situations of concept enforcing (S_2) and concept usages (S_3). The strategy indicates that one of the model elements is discarded and an additional parameter provides the name of the relation between the initial container and the model element that is kept. In FIG. 5, experts have no choice but to add **`PhoneInformation`** to CD_D since it is used by an external service. Therefore they indicate the name of the relation between a **`Witness`** and the new class **`PhoneInformation`**. Similarly, experts relate **`CrisisInformation`** with three properties of the class **`Crisis`**. These properties are replaced by both a new **`CrisisInformation`** class and a relation between **`Crisis`** and **`CrisisInformation`** called **`crisisInfo`**.

6 Automatic Propagation of the Resolution Strategies

The negotiation phase is important for experts to come to an agreement about how to deal with divergences in views. We capture their decisions in a dedicated language that allows automatic propagation across models. Giving a precise interpretation for

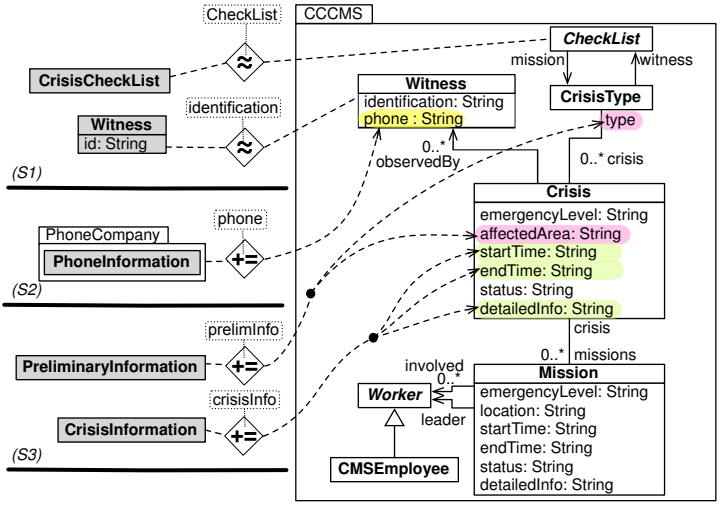


Fig. 5. A mapping model between the extracted model and the domain model is necessary to capture the users expectations

each resolution strategy, we automatically produce a set of operations on both CD_D and BPM to synchronize the views. In the following sections, we illustrate the interpretation of each resolution strategy with examples from the case study.

6.1 Name–Mismatch Strategy

The resolution of name–mismatches is straight-forward. The propagation process identifies every occurrences of a given name and replaces it with the name provided by the experts. The details of the propagation are discussed in the next subsections for both CD_D and BPM.

Domain model synchronization. We use the language of directives provided by the Kompose tool to rename model elements in CD_D. We adapted the Kompose tool to execute directives on a single model. Listing 1.1 lists the directives that the Kompose tool executes for modifying the name of **`CheckList`** in CD_D.

```
Directives{
  domainmodel::CheckList.name := "CrisisCheckList"
}
```

Listing 1.1. Kompose directives for renaming the CheckList class of the domain model CD_D

Business Process Synchronization. We use a formal representation of business processes models, based on many-sorted first order logic [14]. Thus, one can use logical substitution ($\theta = \{x \leftarrow x'\}$, [18]) to replace in a given model m all occurrences of x by x'. We denote a $m\theta$ the model obtained after substitution. When several substitutions $\Theta = \{\theta_1, \ldots, \theta_n\}$ need to be performed on the same model, we denote as $m\Theta$

their parallel application on m. In the context of name mismatch strategies, the engine will generate the set of substitutions necessary to perform all the expected alignments: $\Theta = \{w.identification \leftarrow w.id\}$. Denoting as $\{bp_1, \ldots, bp_n\}$ the available business processes in the system, the enhanced SOA is therefore defined as $\{bp_1\Theta, \ldots, bp_n\Theta\}$.

6.2 Concept Enforcing and Concept Usage Strategies

The resolution of concept enforcing and concept usages situations may rely on a large number of operations for propagating changes. The details of the propagation are discussed in the next subsections for both CD_D and BPM.

Domain Model Synchronization. Synchronization of CD_D for concept enforcing and concept usages relies on a set of Kompose directives to modify CD_D. We adopt two interpretations that are driven by the arity of the mapping relationship:

- When a mapping relationship relates only two model elements, the model element from CD_D is removed, the model element from CD_I is added to CD_D and a UML relation is created from the container of the initial model element from CD_D to the new model element in CD_D. For instance, experts decided to discard the **phone** property of the class **Witness** and use **PhoneInformation** instead. Property **phone** is removed from the class **Witness** and we create a new containment relation between **Witness** and **PhoneInformation**. This relation is named against the parameter of the replacement strategy.
- When a mapping relationship relates more than two model elements, the synchronization process is almost the same except that the model element from CD_I is considered as the container of the model elements from CD_D. Thus, we *move* the model elements from CD_D into the new model element in CD_D. For instance, experts agreed on using **PreliminaryInformation** instead of the two properties **type** and **affectedArea** from the class **Crisis**. **PreliminaryInformation** is thus enriched with the two properties **type** and **affectedArea** and a new containment relation is created between **Crisis** and **PreliminaryInformation**.

Listing 1.2 lists the directives that are applied on CD_D for replacing the **phone** property of the class **Witness** with **PhoneInformation**.

```
Directives{
  /*Creates a new PhoneInformation class
    and removes existing phone attribute
    in Witness*/
  create Class as $pi
  $pi.name = "PhoneInformation"
  destroy domainmodel::Witness::phone
  //Creates the phone relation
  create Association as $phone
  $phone.name = "phone"
  create Property as $phone_src
  $phone_src.aggregation =
   domainmodel::AggregationKind::
    #composite
  $phone_src.upper = 1
  $phone_src.type = domainmodel::Witness

  create Property as $phone_tgt
  $phone_tgt.upper = 1
  $phone_tgt.type = $pi

  $phone.memberEnd + $phone_src
  $phone.memberEnd + $phone_tgt
  /*Adds the PhoneInformation class and
    the phone relation*/
  domainmodel::packagedElement + $pi
  domainmodel::packagedElement + $phone }
```

Listing 1.2. Kompose directives for integrating PhoneInformation in the domain model CD_D

Business Process Synchronization. The propagation of strategies for the resolution of concept enforcing and concept usage situations relies on logical substitution to propagate the new accesses (*e.g.*, $\{pi \leftarrow wi.phone\}$ to replace the variable *pi* by an access to the attribute *phone* contained in the variable *wi*). However, such replacements impose that we retrieve the "container" variable (*e.g.*, *wi*) that is necessary to access a specific property (*e.g.*, *phone*). Synchronization of **`PhoneInformation`** and **`phone`** illustrates the situation where the "container" variable already exists. Thus we use this variable to access to the phone information of a **`Witness`** and substitutions are propagated. When the "container" variable is not already available, we ask the experts how to initialize this "container" in BPM. After synchronization of **`PreliminaryInformation`** with **`type`** and **`affectedArea`**, **`PreliminaryInformation`** is contained by a **`Crisis`** object. Since no **`Crisis`** object is available in the initial process, experts propose the invocation of the **`getCrisis`** operation exposed by the **`CMS`** service. This operation stores a **`Crisis`** object in a variable *c*. This invocation is automatically inserted into the business process by the ADORE engine (after the **`receive`** acitivity) and default substitutions are executed.

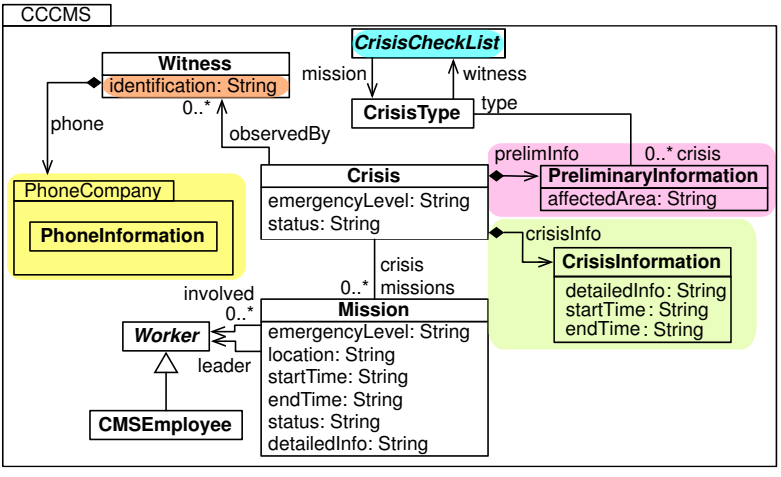
7 Related Work

Researchers and practitioners recognize the importance of business process modeling in understanding and designing accurate software systems [4]. Service-Oriented Architecture supports composition of standard-based services that can be reused quickly to meet business needs. A common enterprise domain model for integration into a SOA is used for exchanging business information between services. A pragmatic approach to support integration of a SOA is to concurrently design the domain model and business processes.

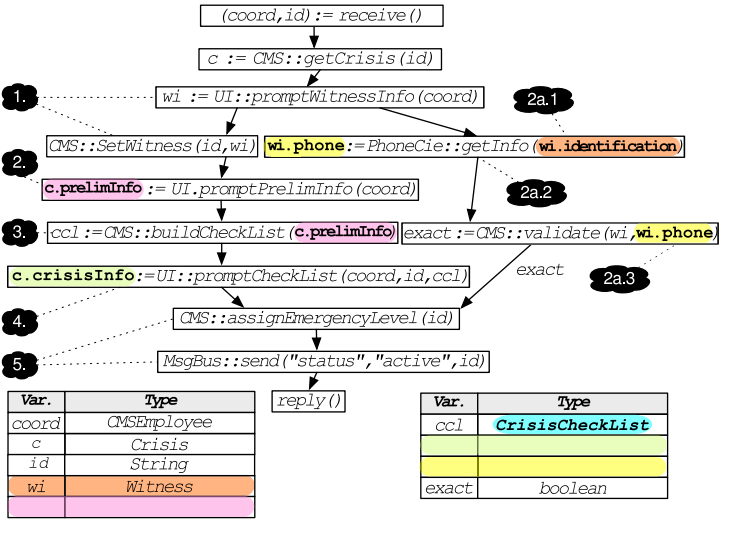
Model matching and model merging are the key activities in most of the multi–modeling approaches that tackle analysis or design of software systems. The techniques for model matching proposed in [16,1,7] are not incompatible with our approach and we may benefit from them to propose a formal basis for model matching. However, this paper focuses on the automation of the divergence detection and of the synchronization process: we propose to capture divergences resolution strategies between heterogeneous domains in a dedicated model and we provide supporting tools for their automatic propagation.

In [3], authors formalize possible conflicts for classes merging. Predefined *Conflict-Fixers* can then be used to automatically solve conflicts. We extend this approach to provide operations that change the business process when necessary.

In [19], the authors extend the UML metamodel to support consistency maintenance between class diagrams, sequence diagrams and state diagrams. We complement this work, focusing on class diagrams and business processes and proposing strategies for resolving differences. In [6], a component modeling language called MiCo has been defined that supports multi-view modeling. The consistency between different diagrams is automatically achieved by building a unique model, gluing the different view models that the users have built. We provide a similar common model but its purpose is to propagate resolution strategies in multiple business processes models.



(a) Aligned domain model



(b) Aligned business process model

Fig. 6. Aligned models, after the synchronization

Among the divergences identified, some require human expertise. Identifying the divergences and proposing changes is similar to refactoring. Kerievsky defines a set of patterns and their corresponding sequences of low-level design transformations, known as refactorings, to improve existing designs [10]. We identify similar patterns for which we propose automatic transformations.

In [13], authors propose the technique of critical pair analysis to detect the implicit dependencies between refactorings. The results of this analysis can help the developer to make an informed decision of which refactoring is most suitable in a given context and why. We are considering integrating this approach with our approach to identifying strategies.

When models of different views are changed, it may be necessary to track these changes. Like [12], we are working to save the changes (synchronization directives) and strategies that have been applied to improve the traceability of the system and automate some particular choice. In the long term we also plan to use this information to allow backtracking and thus support a better management of accidental complexity [2].

8 Conclusion

In this paper we describe an approach for synchronizing business process models with domain models developed by different teams working on the same system. The approach leverages and integrates model composition and generative techniques and tools. While manual intervention is still required, significant aspects of the synchronization process are automated. Manual intervention focuses on activities that require human judgment and experience, for example, on activities concerned with resolving divergences and conflicts across the models. Deciding what to compose and which composition to apply still remains a difficult manual process, due to the many dependencies and interrelationships between relevant compositions.

We plan to dig further for identifying other situations that require specific resolution strategies. Improving the automatic detection of divergences and propose an extensive set of relevant resolution strategies will help managing the global complexity of multi-view synchronization.

Acknowledgments. This work has been partially supported by *(i)* the MOPCOM-I Project from the Images & Réseaux Competitiveness Cluster of Brittany and *(ii)* by the Ministry of Higher Education and Research, Nord–Pas de Calais Regional Council and FEDER through the Contrat de Projets Etat Region Campus Intelligence Ambiante (CPER CIA) 2007-2013.

References

1. Anwar, A., Ebersold, S., Coulette, B., Nassar, M., Kriouile, A.: A Rule-Driven Approach for composing Viewpoint-oriented Models. Journal of Object Technology 9(2), 89–114 (2010)
2. Atkinson, C., Kühne, T.: Reducing accidental complexity in domain models. Software & Systems Modeling 7(3), 345–359 (2007)
3. Barais, O., Klein, J., Baudry, B., Jackson, A., Clarke, S.: Composing Multi-view Aspect Models. In: Seventh International Conference on Composition-Based Software Systems, ICCBSS 2008, pp. 43–52. IEEE, Los Alamitos (2008)

4. Barjis, J.: The importance of business process modeling in software systems design. Science of Computer Programming 71(1), 73–87 (2008)
5. Clavreul, M., Barais, O., Jézéquel, J.M.: Integrating legacy systems with mde. In: ICSE 2010: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering and ICSE Workshops, Cape Town, South Africa, vol. 2, pp. 69–78 (May 2010)
6. De Lara, J., Guerra, E., Vangheluwe, H., de Lara, J., Guerra, E., Vangheluwe, H.: A Multi-View Component Modelling Language for Systems Design: Checking Consistency and Timing Constrains. In: Proceedings of the VMSIS 2005: 2005 Workshop on Visual Modeling for Software Intensive Systems, pp. 27–34 (2005)
7. Falleri, J.-R., Huchard, M., Lafourcade, M., Nebut, C.: Metamodel Matching for Automatic Model Transformation Generation. In: Busch, C., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301, pp. 326–340. Springer, Heidelberg (2008)
8. France, R., Fleurey, F., Reddy, R., Baudry, B., Ghosh, S.: Providing support for model composition in metamodels. In: 11th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2007, p. 253 (October 2007)
9. Katz, S., Mezini, M., Kienzle, J. (eds.): Transactions on Aspect-Oriented Software Development VII. LNCS, vol. 6210. Springer, Heidelberg (2010)
10. Kerievsky, J.: Refactoring to Patterns. Addison-Wesley, Reading (2004)
11. Kienzle, J., Guelfi, N., Mustafiz, S.: Crisis management systems: A case study for aspect-oriented modeling. In: T. Aspect-Oriented Software Development [9], pp. 1–22
12. Mäder, P., Gotel, O., Philippow, I.: Rule-Based Maintenance of Post-Requirements Traceability Relations. In: 16th IEEE International Requirements Engineering, RE 2008, pp. 23–32. IEEE, Los Alamitos (2008)
13. Mens, T., Taentzer, G., Runge, O.: Analysing Refactoring Dependencies Using Graph Transformation. Software and Systems Modeling 6(3), 269–285 (2007)
14. Mosser, S.: Behavioral Compositions in Service-Oriented Architecture. Ph.D. thesis, Université Nice - Sophia Antipolis, ED STIC, Nice, France (October 2010)
15. Mosser, S., Blay-Fornarino, M., France, R.: Workflow design using fragment composition - crisis management system design through adore. In: T. Aspect-Oriented Software Development [9], pp. 200–233
16. Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S., Zave, P.: Matching and Merging of Statecharts Specifications. In: ICSE 2007: Proceedings of the 29th international conference on Software Engineering, pp. 54–64. IEEE Computer Society, Washington, DC (2007)
17. Noy, N.F., Musen, M.A.: Prompt: Algorithm and tool for automated ontology merging and alignment. In: AAAI/IAAI, pp. 450–455 (2000)
18. Stickel, M.E.: A Unification Algorithm for Associative-Commutative Functions. J. ACM 28, 423–434 (1981)
19. Van Der Straeten, R., Mens, T., Simmonds, J., Jonckers, V.: Using description logic to maintain consistency between UML models. In: Stevens, P., Whittle, J., Booch, G. (eds.) UML 2003. LNCS, vol. 2863, pp. 326–340. Springer, Heidelberg (2003)